MEMORIA PRÁCTICA 2

GR2-6

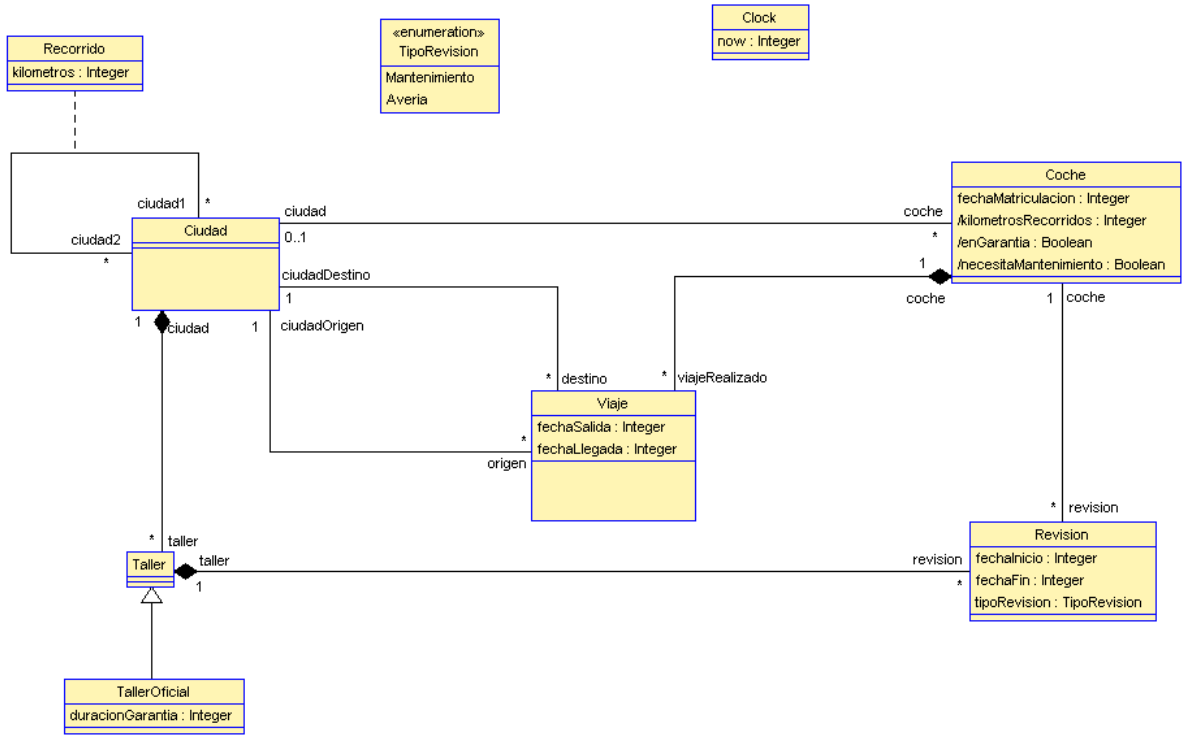
ÍNDICE

1. Diagramas

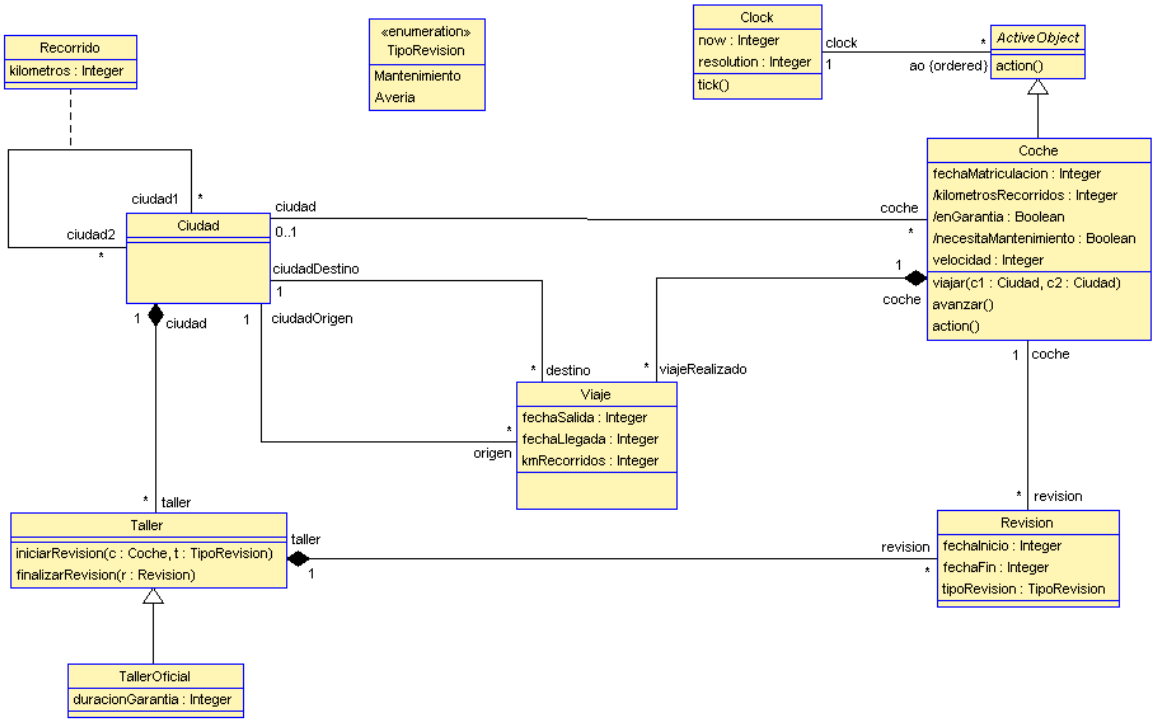
1.1. Diagrama de clases

A continuación, se muestra los diagrama de clases obtenidos en ***USE***:

Apartado a:

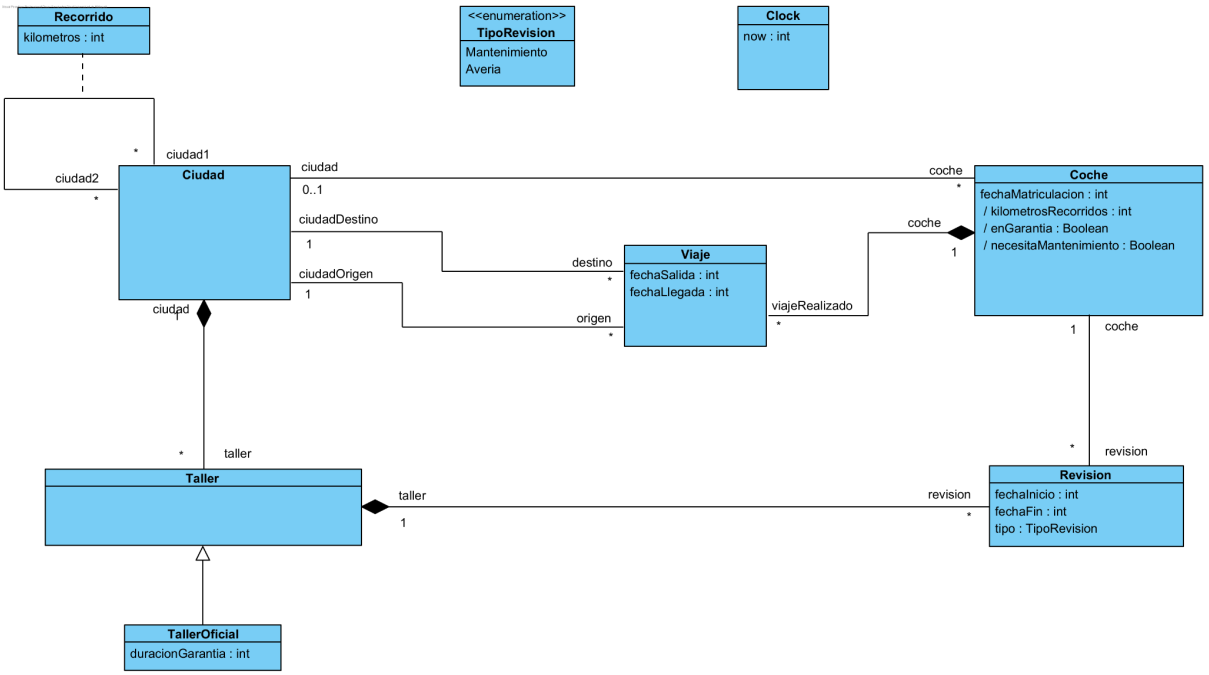


Apartado b:

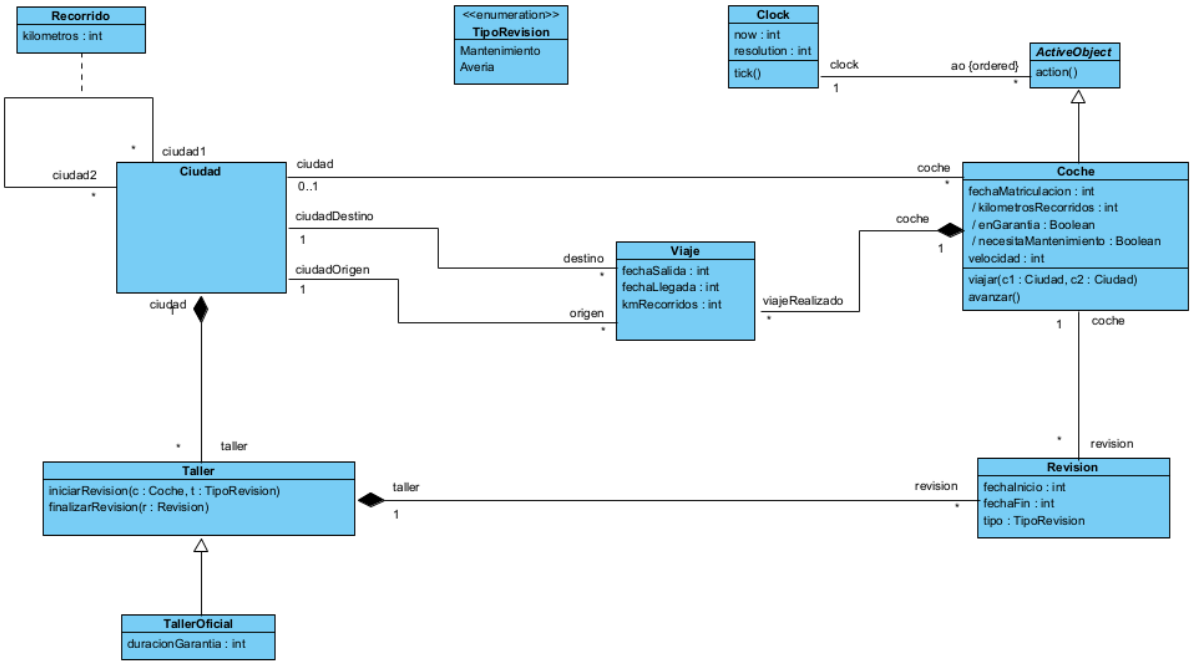


A continuación, se muestran los diagramas de clases obtenido en ***Visual Paradigm***:

Apartado a:



Apartado b:



En el ***diagrama de clases*** expuesto se da solución al problema de sistema de coches planteado. Para ello, contamos con diferentes clases y relaciones que se explicarán a continuación.

En el diagrama podemos identificar las siguientes **entidades o clases**:

- **Coche**: Representa los coches disponibles en el sistema. Sus atributos son "*fechaMatriculacion*" que indica la fecha en la que fue matriculado, "*kilometrosRecorridos*" que refleja el total de kilómetros recorridos, "*enGarantia*" que es un booleano que indica si el vehículo sigue en garantía o no, "*necesitaMantenimiento*" que es otro booleano que indica si necesita mantenimiento o no, y "*velocidad*" que indica la cantidad de kilometros que avanza el coche.

  Las operaciones de esta clase son "viajar" y "avanzar". La primera crea un viaje a partir de dos ciudades. La segunda hace avanzar al coche cada día hasta llegar a la distancia entre las ciudades.

- **Ciudad**: Representa las ciudades del sistema.

- **Taller**: Representa los talleres del sistema. Cuenta con la operación "iniciarRevision", la cual crea una revisión a partir de un coche y una variable "*tipoRevision*". Otra operación es "finalizarRevision" que finaliza la revisión creada anteriormente.

- **TallerOficial**: Representa los talleres oficiales del sistema. Su único atributo es "*duracionGarantia*" que indica el tiempo que dura la garantía.

- **Clock:** Representa el tiempo. Esta clase tiene un atributo "*now*" que indica el día en que nos encontramos. Su operación es "tick" la cual hace que el atributo "*now*" aumente en una unidad.

- **Revisión**: Representa las revisiones que se realizan en los talleres del sistema. Sus atributos son "*fechaInicio*", "*fechaFin*" y "*tipo*" que representan el inicio de la revisión, el fin y el tipo de revisión, es decir, si se trata de un mantenimiento o de una avería.

- **Viaje**: Representa los viajes que se realizan en el sistema. Sus atributos son "*fechaSalida*" y "*fechaLlegada*" e indican cuándo empieza y termina el viaje respectivamente. Además esta clase tiene el atributo "*kmRecorridos*" que indica los kilómetros de distancia recorridos en el viaje.

A su vez también podemos identificar el siguiente enumerator.

- **TipoRevision**: Enumerator que indica los distintos tipos de revisión que existen; "*Mantenimiento*" y "*Avería*".

Por otro lado, encontramos la siguiente ***clase de asociación***:

- **Recorrido**: Clase de asociación que sirve para conectar dos ciudades. Su único atributo es "*kilometros*" e indica la distancia entre las dos ciudades.

Por último, podemos destacar las siguientes ***relaciones*** que permiten asociar las diferentes entidades anteriormente mencionadas.

- Como ya se explicó anteriormente, "*Taller*" actúa como clase base para *"TallerOficial"* por lo que tiene una relación de herencia con esta clase.

- *"Ciudad"* tiene una relación de asociación consigo misma. La cardinalidad de esta asociación es de *"*"* a *"*",* indicando que muchas ciudades pueden estar relacionadas con muchas otras. Esta relación cuenta con la clase de asociación llamada "*Recorrido*".

- "*Ciudad*" tiene dos clases de asociación con "*Viaje*". La asociación llamada "DestinoViaje" tiene una cardinalidad de "*1*" en "*Ciudad*" y de "** en "Viaje",* indicando que una ciudad puede ser la ciudad destino de muchos viajes. *La asociación "*OrigenViaje*" también tiene una asociación de "1" en "Ciudad" y "**" en "*Viaje*" indicando que una ciudad puede ser el origen de muchos viajes.

- "*Ciudad*" cuenta con una asociación con "*Coche*" con una cardinalidad de "*0…1*" a "***" respectivamente, lo que indica que un coche puede estar como máximo en una ciudad, y que muchos coches pueden estar en una ciudad.

- "*Viaje*" está asociado con "*Coche*" con una relación llamada "*Viajar*" cuya cardinalidad es "***" por parte de "*Viaje*" y de "*1*" por parte de "*Coche*" indicando que un coche puede realizar muchos viajes pero que un viaje sólo es realizado por un coche.

- "*Coche*" está relacionado con "*Revisión*" por medio de una asociación llamada "CocheRevision" cuya cardinalidad es de "*1*" a "***" respectivamente, por lo que un coche puede ir a múltiples revisiones, pero en cada revisión sólo puede haber un coche.

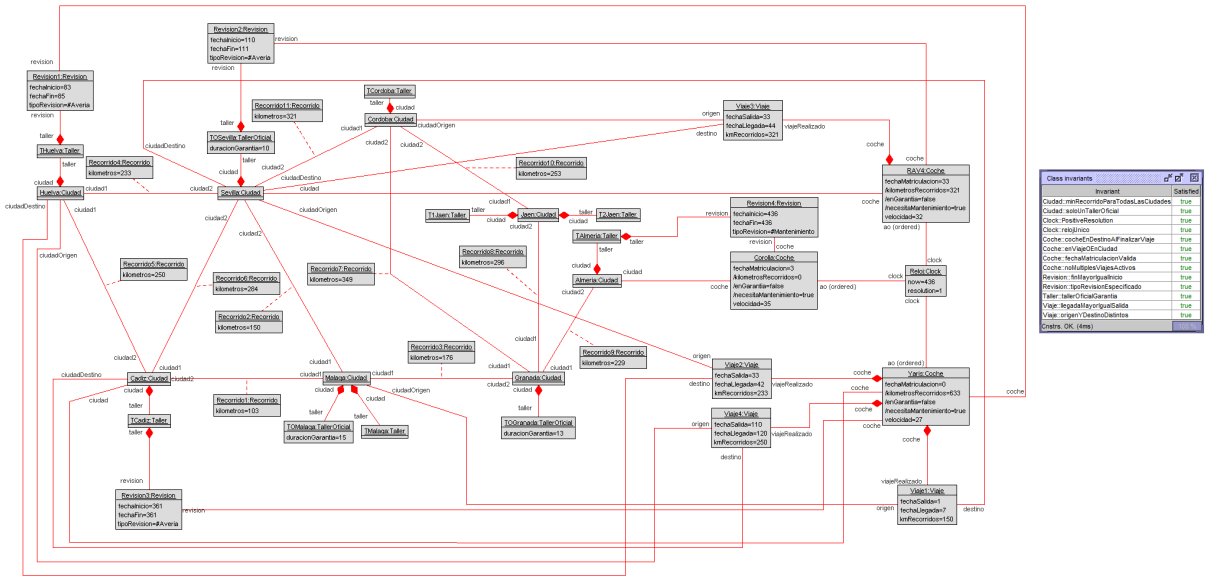
- "*Revision*" tiene una relación de composición con "*Taller*" cuya cardinalidad es de "***" a "*1*" respectivamente, indicando que un taller puede tener muchas revisiones, pero una revisión solo se hace en un taller.

- Por último, "*Taller*" tiene una relación de composición con "*Ciudad*" cuya multiplicidad es de "***" a "*1*" respectivamente, por lo que una ciudad puede tener muchos talleres, pero un taller sólo puede estar en una ciudad.

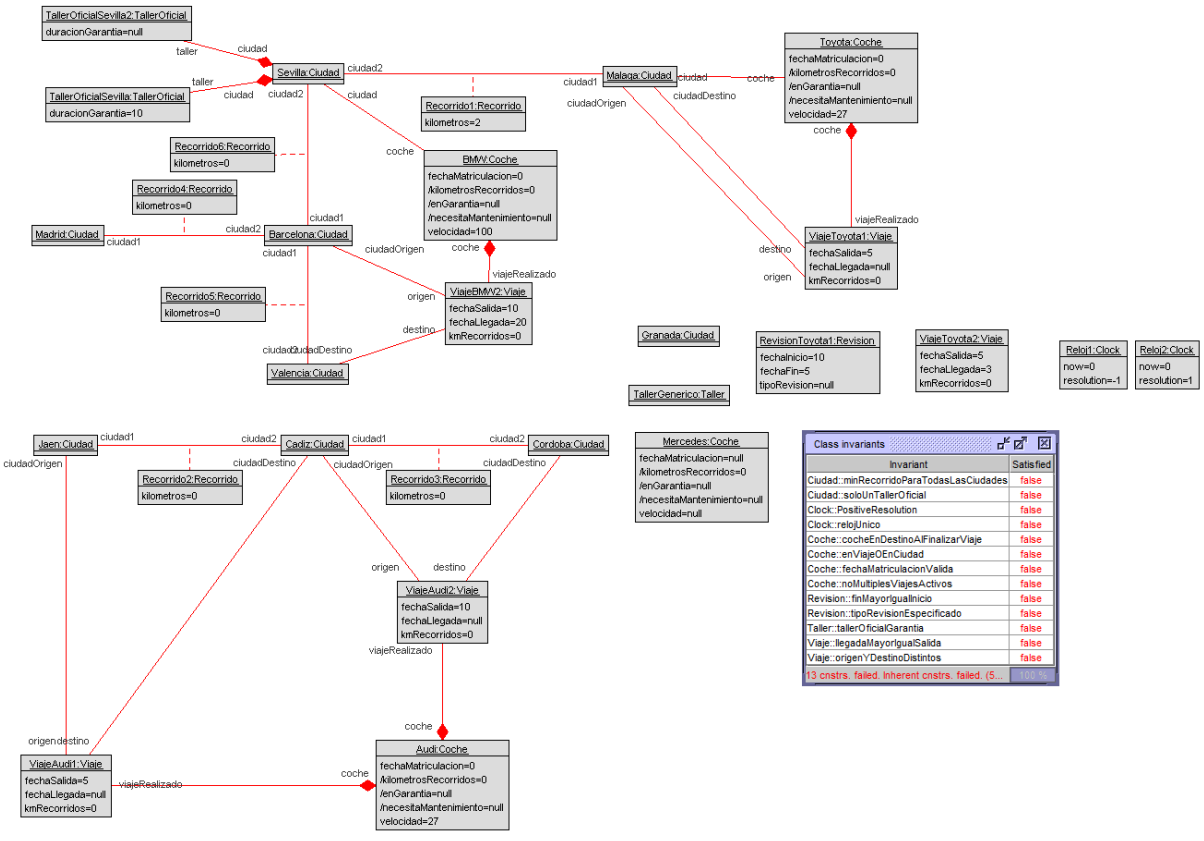
1.2. Diagramas de objetos

1.2.1. Apartado a+b

1.2.1.1. Diagrama P2.soil



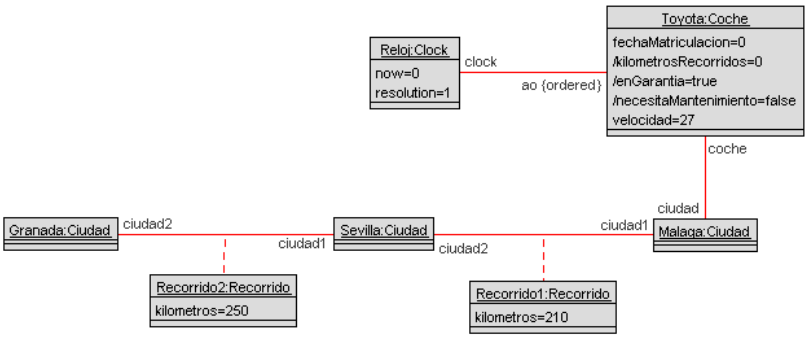
1.2.1.2. Diagrama P2Fallo.soil

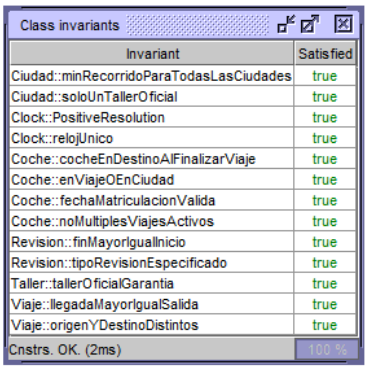


1.2.2. Apartado c

Instante 0:

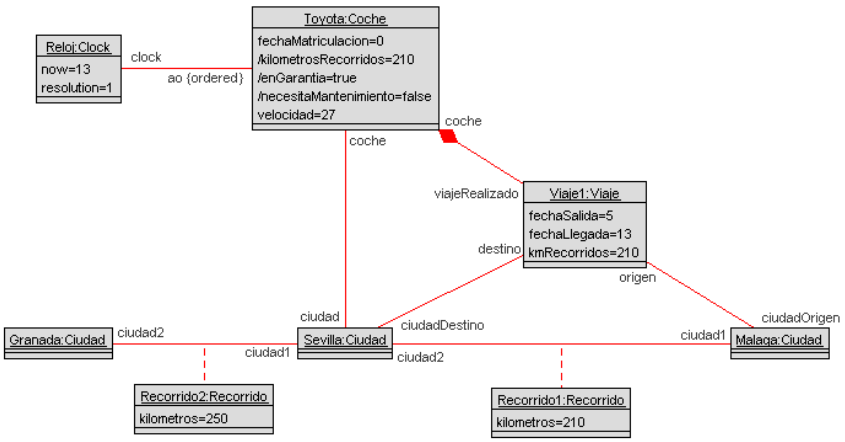
En este instante el coche se encuentra en la ciudad de Málaga a 210 kilómetros de Sevilla y cuenta con 0 kilómetros recorridos (Día 5).

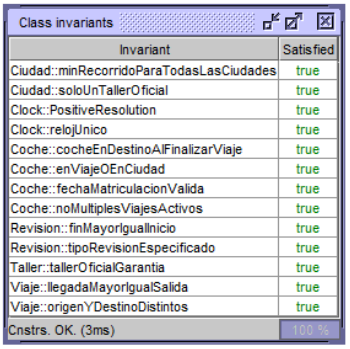




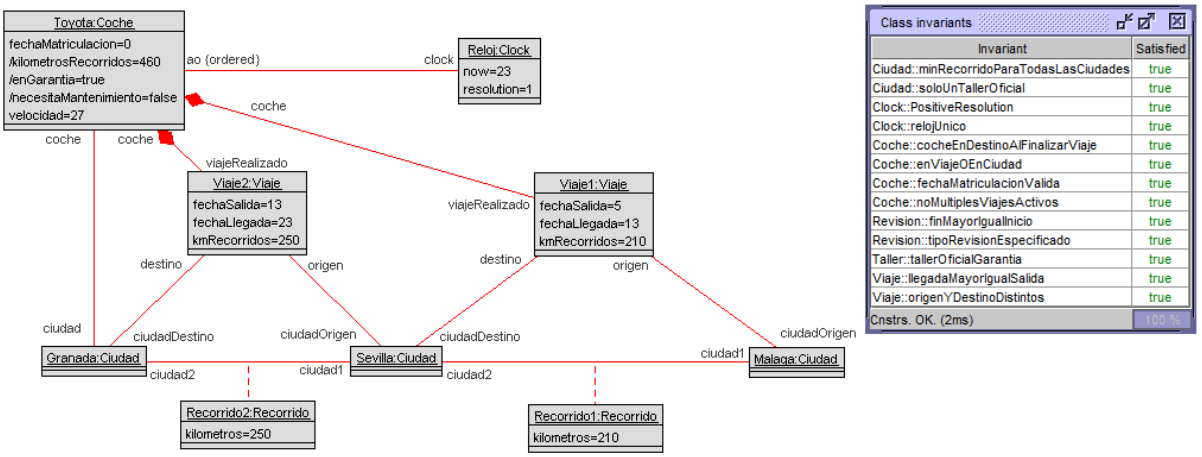
Llegada a Sevilla:

En este instante el coche ha llegado a la ciudad intermedia de Sevilla (Día 12) con 210 kilómetros recorridos, se ha creado la instancia Viaje con sus respectivas relaciones (Ciudad Origen, Ciudad Destino) y le restan 250 kilómetros hasta la ciudad de destino final Granada.





Llegada a Granada:
En este instante final el coche ya ha recorrido 460 kilómetros, ha llegado al destino final Granada en el día 22.



2. Invariantes o Restricciones OCL

A continuación, se muestran las diferentes ***invariantes*** que hemos establecido junto a una breve descripción:

```
constraints

context Viaje
    -- Un viaje debe tener una ciudad de origen y una de destino distintas entre sí.
    inv origenYDestinoDistintos: self.ciudadDestino <> self.ciudadOrigen

    -- Dos viajes no pueden solaparse en el tiempo, es decir, un viaje debe ocurrir siempre después de otro.
    inv llegadaMayorIgualSalida: if self.fechaLlegada <> null then self.fechaLlegada >= self.fechaSalida else true endif

context Revision
    -- La fecha de fin de la revisión debe ser posterior o igual a la fecha de inicio.
    inv finMayorIgualInicio: if self.fechaFin <> null then self.fechaFin >= self.fechaInicio else true endif

    -- Cada revisión debe tener un tipo de revisión.
    inv tipoRevisionEspecificado: self.tipoRevision = #Mantenimiento or self.tipoRevision = #Averia

context Ciudad
    -- Cada ciudad debe estar al menos a 5 kilómetros de recorrido de otra ciudad.
    inv minRecorridoParaTodasLasCiudades: Ciudad.allInstances() -> forAll(c1 | Ciudad.allInstances() -> exists(c2 | c1 <> c2 and Recorrido.allInstances() ->
        exists(d |(d.ciudad1 = c1 and d.ciudad2 = c2 or d.ciudad1 = c2 and d.ciudad2 = c1) and d.kilometros >= 5)))

    -- En una ciudad solo puede haber un taller oficial.
    inv soloUnTallerOficial: self.taller -> select(t | t.oclIsTypeOf(TallerOficial)) -> size() <= 1

context Coche
    -- Un coche se encontrará en todo momento bien realizando un viaje determinado o bien en una ciudad.
    inv enViajeOEnCiudad: (self.viajeRealizado -> exists(v | v.fechaLlegada = null)) xor (self.ciudad -> notEmpty())
        --xor : nos asegura que solo una de estas condiciones sea verdadera a la vez

    -- Al finalizar un viaje el coche se encuentra en la ciudad destino.
    inv cocheEnDestinoAlFinalizarViaje: self.viajeRealizado -> exists(v | v.fechaLlegada <> null) implies
        (self.ciudad = self.viajeRealizado -> select(v | v.fechaLlegada <> null) -> asSequence() -> last().ciudadDestino)

    -- Un coche no puede realizar más de un viaje a la vez.
    inv noMultiplesViajesActivos: self.viajeRealizado -> select(v | v.fechaLlegada = null) -> size() <= 1

    -- La fecha de matriculación debe ser válida (positiva y menor o igual que la del reloj).
    inv fechaMatriculacionValida: self.fechaMatriculacion >= 0 and self.fechaMatriculacion <= Clock.allInstances() -> asOrderedSet() -> first().now

context Taller
    -- Un taller oficial debe tener siempre una duracion de garantia distinta de null.
    inv tallerOficialGarantia: Taller.allInstances() -> select(t | t.oclIsKindOf(TallerOficial)) -> forAll(t | t.oclAsType(TallerOficial).duracionGarantia <> null)

context Clock
    -- Existe una única instancia del reloj.
    inv relojUnico: Clock.allInstances() -> size() = 1

    -- El número de días que pasa al llamar a la operación tick del reloj es positivo.
    inv PositiveResolution: self.resolution > 0
```